Learning Pseudo-Boolean k-DNF and Submodular Functions

Sofya Raskhodnikova*
Pennsylvania State University
sofya@cse.psu.edu

Grigory Yaroslavtsev†
Pennsylvania State University
grigory@cse.psu.edu

October 8, 2012

Abstract

We prove that any submodular function $f : \{0,1\}^n \to \{0,1,...,k\}$ can be represented as a pseudo-Boolean $2k$-DNF formula. Pseudo-Boolean DNFs are a natural generalization of DNF representation for functions with integer range. Each term in such a formula has an associated integral constant. We show that an analog of Håstad's switching lemma holds for pseudo-Boolean k-DNFs if all constants associated with the terms of the formula are bounded.

This allows us to generalize Mansour's PAC-learning algorithm for k-DNFs to pseudo-Boolean k-DNFs, and hence gives a PAC-learning algorithm with membership queries under the uniform distribution for submodular functions of the form $f : \{0,1\}^n \to \{0,1,...,k\}$. Our algorithm runs in time polynomial in n, $k^{O(k \log k/\epsilon)}$ and $\log(1/\delta)$ and works even in the agnostic setting. The line of previous work on learning submodular functions [Balcan, Harvey (STOC '11), Gupta, Hardt, Roth, Ullman (STOC '11), Cheraghchi, Klivans, Kothari, Lee (SODA '12)] implies only $n^{O(k)}$ query complexity for learning submodular functions in this setting, for fixed ϵ and δ.

Our learning algorithm implies a property tester for submodularity of functions $f : \{0,1\}^n \to \{0,\ldots,k\}$ with query complexity polynomial in n for $k = O((\log n/\log\log n)^{1/2})$ and constant proximity parameter ϵ.

*Supported by NSF CAREER award CCF-0845701.

†Supported by College of Engineering Fellowship at Pennsylvania State University and NSF CAREER award CCF-0845701.

1 Introduction

We investigate learning of submodular set functions, defined on the ground set $[n] = \{1,\ldots,n\}$. A set function $f : 2^{[n]} \to \mathbb{R}$ is *submodular* if one of the following equivalent definitions holds:

Definition 1.1. *1. For all $S,T \subseteq [n]$:*

$$f(S) + f(T) \geq f(S \cup T) + f(S \cap T).$$

2. For all $S \subset T \subseteq [n]$ and $i \in [n] \setminus T$:

$$f(S \cup \{i\}) - f(S) \geq f(T \cup \{i\}) - f(T).$$

3. For all $S \subseteq [n]$ and $i,j \in [n] \setminus S$:

$$f(S \cup \{i\}) + f(S \cup \{j\}) \geq f(S \cup \{i,j\}) + f(S).$$

Submodular set functions are important and widely studied, with applications in combinatorial optimization, economics, algorithmic game theory and many other disciplines. In many contexts, submodular functions are integral and nonnegative, and this is the setting we focus on. Examples of such functions include coverage functions[1], matroid rank functions, functions modeling valuations when the value of each set is expressed in dollars, cut functions of graphs[2], and cardinality-based set functions,

[1]Given sets $A_1,\ldots,A_n$ in the universe U, a coverage function is $f(S) = |\cup_{j\in S} A_j|$.

[2]Given a graph G on the vertex set $[n]$, the cut function $f(S)$ of G is the number of edges of G crossing the cut $(S,[n]/S)$).

i.e., functions of the form $f(S) = g(|S|)$, where g is concave.

We study submodular functions $f : 2^{[n]} \to \{0, 1, \ldots, k\}$, and give a learning algorithm for this class. To obtain our result, we use tools from several diverse areas, ranging from operations research to complexity theory.

Structural result. The first ingredient in the design of our algorithm is a structural result which shows that every submodular function in this class can be represented by a narrow pseudo-Boolean disjunctive normal form (DNF) formula, which naturally generalizes DNF for pseudo-Boolean functions. Pseudo-Boolean DNF formulas are well studied. (For an introduction to pseudo-Boolean functions and normal forms, see §13 of the book by Crama and Hammer [11].)

In the next definition and the rest of the paper, we use domains $2^{[n]}$ and $\{0,1\}^n$ interchangeably. They are equivalent because there is a bijection between sets $S \subseteq [n]$ and strings $x_1 \ldots x_n \in \{0,1\}^n$, where the bit x_i is mapped to 1 if $i \in S$ and to 0 otherwise.

Definition 1.2 (Pseudo-Boolean DNF)**.** *Let $x_1, \ldots, x_n$ be variables taking values in $\{0,1\}$. A* pseudo-boolean DNF of width k and size s *(also called a k-DNF of size s) is an expression of the form*

$$f(x_1, \ldots, x_n) = \max_{t=1}^{s} \Big(a_t \Big(\bigwedge_{i \in A_t} x_i \Big) \Big(\bigwedge_{j \in B_t} \bar{x}_j \Big) \Big),$$

where a_t are constants, $A_t, B_t \subseteq [n]$ and $|A_t| + |B_t| \leq k$ for $t \in [s]$. A pseudo-boolean DNF is monotone *if it contains no negated variables, i.e., $B_t = \emptyset$ for all* terms *in the* max *expression. The class of all functions that have pseudo-Boolean k-DNF representations with constants $a_t \in \{0, \ldots r\}$ is denoted $DNF^{k,r}$.*

It is not hard to see that every set function $f : 2^{[n]} \to \{0, \ldots, k\}$ has a pseudo-Boolean DNF representation with constants $a_t \in \{0, \ldots, k\}$, but in general there is no bound on the width of the formula.

Our structural result, stated next, shows that every submodular function $f : 2^{[n]} \to \{0, \ldots, k\}$ can be represented by a pseudo-Boolean $2k$-DNF with constants $a_t \in \{0, \ldots, k\}$. Our result is stronger for *monotone* functions, i.e., functions satisfying $f(S) \leq f(T)$ for all $S \subseteq T \subseteq [n]$. Examples of monotone submodular functions include coverage functions and matroid rank functions.

Theorem 1.1 (DNF representation of submodular functions)**.** *Each submodular function $f : \{0,1\}^n \to \{0, \ldots, k\}$ can be represented by a pseudo-Boolean $2k$-DNF with constants $a_t \in \{0, \ldots, k\}$ for all $t \in [s]$. Moreover, each term of the pseudo-Boolean DNF has at most k positive and at most k negated variables, i.e., $|A_t| \leq k$ and $|B_t| \leq k$. If f is monotone then its representation is a monotone pseudo-Boolean k-DNF.*

Note that the converse of Theorem 1.1 is false. E.g., consider the function $f(S)$ that maps S to 0 if $|S| \leq 1$ and to 1 otherwise. It can be represented by a 2-DNF as follows: $f(x_1 \ldots x_n) = \max_{i,j \in [n]} x_i \wedge x_j$. However, it is not submodular, since version 3 of the definition above is falsified with $S = \emptyset, i = 1$ and $j = 2$.

Our proof of Theorem 1.1 builds on techniques developed by Gupta *et al.* [17] who show how to decompose a given submodular function into Lipschitz submodular functions. We first prove our structural result for monotone submodular functions. We use the decomposition from [17] to cover the domain of such a function by regions where the function is constant and then capture each region by a monotone term of width at most k. Then we decompose a general submodular function f into monotone regions, as in [17]. For each such region, we construct a monotone function which coincides with f on that region, does not exceed f everywhere else, and can be represented as a narrow pseudo-Boolean k-DNF by invoking our structural result for monotone submodular functions. This construction uses a monotone extension of submodular functions defined by Lovasz [21].

Learning. Our main result is a PAC-learning algorithm with membership queries under the uniform distribution for pseudo-Boolean k-DNF, which by Theorem 1.1 also applies to submodular functions $f : 2^{[n]} \to \{0, \ldots, k\}$. We use a (standard) variant of the PAC-learning definition given by Valiant [29].

Definition 1.3 (PAC and agnostic learning under uniform distribution)**.** *Let U^n be the uniform distribution on $\{0,1\}^n$. A class of functions $\mathcal{C}$ is* PAC-learnable *under the uniform distribution if there exists a randomized algorithm $\mathcal{A}$, called a* PAC-learner*, which for every function $f \in \mathcal{C}$ and every $\epsilon, \delta > 0$, with probability at least $1-\delta$ over the randomness of $\mathcal{A}$, outputs a hypothesis h, such that*

$$\Pr_{x \sim U^n}[h(x) \neq f(x)] \leq \epsilon. \tag{1}$$

A learning algorithm $\mathcal{A}$ is proper *if it always outputs a hypothesis h from the class $\mathcal{C}$. A learning algorithm is* agnostic *if it works even if the input function f is arbitrary (not necessarily from $\mathcal{C}$), with ϵ replaced by $opt + \epsilon$ in (1), where opt is the smallest achievable error for a hypothesis in $\mathcal{C}$.*

Our algorithm accesses its input f via *membership queries*, i.e., by requesting $f(x)$ on some x in f's domain.

Theorem 1.2. *The class of pseudo-Boolean k-DNF formulas on n variables with constants in the range $\{0,\dots,r\}$ is PAC-learnable with membership queries under the uniform distribution with running time polynomial in n, $k^{O(k \log r/\epsilon)}$ and $\log(1/\delta)$, even in the agnostic setting.*

Our (non-agnostic) learning algorithm is a generalization of Mansour's PAC-learner for k-DNF [22]. It consists of running the algorithm of Kushilevitz and Mansour [19] for learning functions that can be approximated by functions with few non-zero Fourier coefficients, and thus has the same running time (and the same low-degree polynomial dependence on n). To be able to use this algorithm, we prove (in Theorem 4.1) that all functions in $\mathrm{DNF}^{k,r}$ have this property. The agnostic version of our algorithm follows from the Fourier concentration result in Theorem 4.1 and the work of Gopalan, Kalai and Klivans [16].

The key ingredient in the proof of Theorem 4.1 (on Fourier concentration) is a generalization of Håstad's switching lemma [18, 5] for standard DNF formulas to pseudo-Boolean DNF. Our generalization (formally stated in Lemma 3.1) asserts that a function $f \in \mathrm{DNF}^{k,r}$, restricted on large random subset of variables to random Boolean values, with large probability can be computed by a decision tree of small depth. (See Section 3 for definitions of random restrictions and decision trees.) Crucially, our bound on the probability that a random restriction of f has large decision-tree complexity is only a factor of r larger than the corresponding guarantee for the Boolean case.

Theorems 1.2 and 1.1 imply the following corollary.

Corollary 1.3. *The class of submodular functions $f\colon \{0,1\}^n \to \{0,\dots,k\}$ is PAC-learnable with membership queries under the uniform distribution in time polynomial in n, $k^{O(k \log k/\epsilon)}$ and $\log(1/\delta)$, even in the agnostic setting.*

Implications for testing submodularity. Our results give property testers for submodularity of functions $f : 2^{[n]} \to \{0,\dots,k\}$. A *property tester* [26, 14] is given oracle access to an object and a proximity parameter $\epsilon \in (0,1)$. If the object has the desired property, the tester *accepts* it with probability at least $2/3$; if the object is ϵ-far from having the desired property then the tester *rejects* it with probability at least $2/3$. Specifically, for properties of functions, ϵ-far means that a given function differs on at least an ϵ fraction of the domain points from any function with the property.

As we observe in Proposition A.1, a learner for a discrete class (e.g., the class of functions $f : 2^{[n]} \to \{0,\dots,k\}$) can be converted to a proper learner with the same query complexity (but huge overhead in running time). Thus, Corollary 1.3 implies a tester for submodularity of functions $f : 2^{[n]} \to \{0,\dots,k\}$ with query complexity polynomial in n and $k^{O(k \log k/\epsilon)}$, making progress on a question posed by Seshadhri [27].

1.1 Related work

Structural results for Boolean submodular functions. For the special case of Boolean functions, characterizations of submodular and monotone submodular functions in terms of simple DNF formulas are known. A Boolean function is monotone submodular if and only if it can be represented as a

monotone 1-DNF (see, e.g., Appendix A in [4]). A Boolean function is submodular if and only if it can be represented as $\left(\bigvee_{i\in S} x_i\right) \wedge \left(\bigvee_{j\in T} \bar{x}_j\right)$ for $S, T \subseteq [n]$ [12].

Learning submodular functions. The problem of learning submodular functions has recently attracted significant interest. The focus on learning-style guarantees, which allow one to make arbitrary mistakes on some small portion of the domain, is justified by a negative results of Goemans *et al.* [13]. It demonstrates that every algorithm that makes a polynomial in n number of queries to a monotone submodular function (more specifically, even a matroid rank function) and tries to approximate it on all points in the domain, must make an $\Omega(\sqrt{n}/\log n)$ multiplicative error on some point.

Using results on concentration of Lipschitz submodular functions [7, 8, 30] and on noise-stability of submodular functions [10], significant progress on learning submodular functions was obtained by Balcan and Harvey [4, 3], Gupta *et al.* [17] and Cheraghchi *et al.* [10]. These works obtain learners that *approximate* submodular functions, as opposed to learning them exactly, on an ϵ fraction of values in the domain. However, their learning algorithms generally work with weaker access models and with submodular functions over more general ranges.

Balcan and Harvey's algorithms learn a function within a given *multiplicative error* on all but ϵ fraction of the probability mass (according to a specified distribution on the domain). Their first algorithm learns *monotone* nonnegative submodular functions over $2^{[n]}$ within a *multiplicative factor of* $\sqrt{n}$ over *arbitrary* distributions using only *random examples* in polynomial time. For the special case of *product distributions* and *monotone* nonnegative submodular functions *with Lipschitz constant 1*, their second algorithm can learn *within a constant factor* in polynomial time.

Gupta *et al.* [17] design an algorithm that learns a submodular function with the range $[0, 1]$ within a given *additive error* α on all but ϵ fraction of the probability mass (according to a specified *product distribution* on the domain). Their algorithm requires membership queries, but works *even when these queries are answered with additive error* $\alpha/4$. It takes $n^{O(\log(1/\epsilon)/\alpha^2)}$ time.

Cheraghchi *et al.* [10] also work with *additive error*. Their learner is agnostic and only uses *statistical queries*. It produces a hypothesis which (with probability at least $1 - \delta$) has the expected additive error $opt + \alpha$ with respect to a *product distribution*, where *opt* is the error of the best concept in the class. Their algorithm runs in time polynomial in $n^{O(1/\alpha)}$ and $\log(1/\delta)$.

Observe that the results in [17] and [10] directly imply an $n^{O(\log(1/\epsilon)k^2)}$ time algorithm for our setting, by rescaling our input function to be in $[0, 1]$ and setting the error $\alpha = 1/(2r)$. The techniques in [17] also imply $n^{O(k)}$ time complexity for non-agnostically learning submodular functions in this setting, for fixed ϵ and δ. To the best of our knowledge, this is the best dependence on n, one can obtain from previous work.

Chakrabarty and Huang [9] gave an exact learning algorithm for coverage functions, a subclass of monotone submodular functions. Their algorithm makes $O(n|U|)$ queries, where U is the size of the universe. (Coverage functions are defined as in Footnote 1 with additional nonnegative weight for each set, and $f(S)$ equal to the weight of $\cup_{j\in S} A_j$ instead of the cardinality.)

In a related line of work, focused on learning subadditive and fractionally subadditive functions with multiplicative approximation positive results are obtained by Balcan, Constantin, Iwata and Wang [2] and by Badanidiyuru, Dobzinski, Fu, Kleinberg, Nisan and Roughgarden [1].

Property testing submodular functions. The study of submodularity in the context of property testing was initiated by Parnas, Ron and Rubinfeld [24]. Seshadhri and Vondrak [28] gave the first sublinear (in the size of the domain) tester for submodularity of set functions. Their tester works for all ranges and has query and time complexity $(1/\epsilon)^{O(\sqrt{n}\log n)}$. They also showed a reduction from testing monotonicity to testing submodularity which, together with a lower bound for testing monotonicity

given by Blais, Brody and Matulef [6], implies a lower bound of $\Omega(n)$ on the query complexity of testing submodularity for an arbitrary range and constant $\epsilon > 0$.

Given the large gap between known upper and lower bounds on the complexity of testing submodularity, Seshadhri [27] asked for testers for several important subclasses of submodular functions. The exact learner of Chakrabarty and Huang [9] for coverage functions, mentioned above, gives a property tester for this class with the same query complexity.

For the special case of Boolean functions, in the light of the structural results mentioned above, one can test if a function is monotone submodular with $O(1/\epsilon)$ queries by using the algorithm from [25] (Section 4.3) for testing monotone monomials.

2 Structural result

In this section, we prove Theorem 1.1 that shows that every submodular function over a bounded (nonnegative) integral range can be represented by a narrow pseudo-Boolean DNF. After introducing notation used in the rest of the section (in Definition 2.1), we prove the theorem for the special case when f is monotone submodular (restated in Lemma 2.1) and then present the proof for the general case. In the proof, we give a recursive algorithm for constructing pseudo-Boolean DNF representation which has the same structure of recursive calls as the decomposition algorithm of Gupta *et al.* [17] for the monotone case. Our contribution is in showing how to use these calls to get a monotone pseudo-Boolean k-DNF representation of the input function. For the non-monotone case the structure of recursive calls that we use is different from that of [17].

Lemma 2.1 (DNF representation of monotone submodular functions)**.** *Every monotone submodular function $f : \{0,1\}^n \to \{0,\dots,k\}$ can be represented by a pseudo-Boolean monotone k-DNF with constants $a_t \in \{0,\dots,k\}$ for all $t \in [s]$.*

Proof. Algorithm 1 below, with the initial call MONOTONE-DNF$(f, \emptyset)$, returns the collection $\mathcal{C}$ of terms in a pseudo-boolean DNF representation of f.

Algorithm 1: MONOTONE-DNF(f, S).

input : Oracle access to $f : 2^{[n]} \to \{0,\dots,k\}$, argument $S \in 2^{[n]}$.
output: Collection $\mathcal{C}$ of monotone terms of width at most k.

```
C ← (f(S) · ⋀_{i∈S} x_i)
for j ∈ [n] \ S do
    if f(S ∪ {j}) > f(S) then
        C ← C ∪ MONOTONE-DNF(f, S ∪ {j}).
return C
```

First, note that the invariant $f(S) \geq |S|$ is maintained for every call MONOTONE-DNF(f, S). Since the maximum value of f is at most k, there are no calls with $|S| > k$. Thus, every term in the collection returned by MONOTONE-DNF$(f, \emptyset)$ has width at most k. By definition, all terms are monotone.

Next, we show that the resulting formula $\max_{C_i \in \mathcal{C}} C_i$ exactly represents f. For a clause $C_i \in \mathcal{C}$ and $S \in 2^{[n]}$ we use an indicator function $C_i(S) \to \{0,1\}$ such that $C_i(S) = 1$ iff C_i is satisfied by the assignment of values to its variables according to S. For all $Y \in 2^{[n]}$ we have $f(Y) \geq \max_{C_i \in \mathcal{C}} C_i(Y)$ by monotonicity of f. To see that for all Y we have $f(Y) \leq \max_{C_i \in \mathcal{C}} C_i(Y)$, fix an arbitrary Y and let $\mathcal{T} = \{Z \mid Z \subseteq Y, f(Z) = f(Y)\}$ and T be a set of the smallest size in $\mathcal{T}$. If there was a recursive call MONOTONE-DNF(f, T) then the term added by this recursive call would ensure the inequality. If $T = \emptyset$ then such a call was made. Otherwise, consider the set $\mathcal{U} = \{T \setminus \{j\} \mid j \in T\}$. By the choice of T, we have $f(Z) < f(T)$ for all $Z \in \mathcal{U}$. By submodularity of f (see Definition 1.1, part 2), this implies that the restriction of f on $T^{\downarrow}$ is a strictly increasing function. Thus, the recursive call MONOTONE-DNF(f, T) was made and the term added by it guarantees the inequality. □

For a collection $\mathcal{S}$ of subsets of $[n]$, let $f_{\mathcal{S}} \colon \mathcal{S} \to \mathbb{R}$ denote the restriction of a function f to the union of sets in $\mathcal{S}$. We use notation $\mathbf{1}_{\mathcal{S}} \colon 2^{[n]} \to \{0,1\}$ for the indicator function defined by $\mathbf{1}_{\mathcal{S}}(Y) = 1$ iff $Y \in$

$\bigcup_{S\in\mathcal{S}} S$.

Definition 2.1 ($S^{\downarrow}$ and $S^{\uparrow}$)**.** *For a set $S \in 2^{[n]}$, we denote the collection of all subsets of S by $S^{\downarrow}$ and the collection of all supersets of S by $S^{\uparrow}$.*

Proof of Theorem 1.1. For a general submodular function, the formula can be constructed using Algorithm 2, with the initial call DNF$(f,[n])$. The algorithm description uses the function $f^{mon}_{S^{\downarrow}}$, defined next.

Definition 2.2 (Function $f^{mon}_{S^{\downarrow}}$)**.** *For a set $S \subseteq [n]$, define the function $f^{mon}_{S^{\downarrow}}\colon S^{\downarrow} \to \{0,\ldots,k\}$ as follows: $f^{mon}_{S^{\downarrow}}(Y) = \min_{Y\subseteq Z\subseteq S} f(Z)$.*

Note that if f is a submodular function then for every set $S \subseteq [n]$ the function $f_{S^{\downarrow}}$ is a submodular function.

Proposition 2.2 (Proposition 2.1 in [21])**.** *For every set $S \subseteq [n]$, if $f_{S^{\downarrow}}$ is a submodular function, then $f^{mon}_{S^{\downarrow}}$ is a monotone submodular function.*

Algorithm 2: DNF(f,S).

input : Oracle access to $f : 2^{[n]} \to \{0,\ldots,k\}$, argument $S \in 2^{[n]}$.

output: Collection $\mathcal{C}$ of terms, each containing at most k positive and at most k negated variables.

1 $C_{mon} \leftarrow$ MONOTONE-DNF$(f^{mon}_{S^{\downarrow}}, \emptyset)$

2 $C \leftarrow \bigcup_{C_i\in C_{mon}} (C_i \cdot (\bigwedge_{i\in[n]\setminus S} \bar{x}_i))$

3 **for** $j \in S$ **do**

4 **if** $f(S\setminus\{j\}) > f(S)$ **then**

5 $C \leftarrow C \cup$ DNF$(f, S\setminus\{j\})$.

6 **return** C

Let $\mathcal{S}$ be the collection of sets $S \subseteq [n]$ for which a recursive call is made when DNF$(f,[n])$ is executed. For a set $S \in \mathcal{S}$, let $B(S) = \{j \mid f(S\setminus\{j\}) \le f(S)\}$ be the set consisting of elements such that if we remove them from S, the value of the function does not increase. Let the *monotone region* of S be defined by $S^{\le\downarrow} = \{Z \mid (S\setminus B(S)) \subseteq Z \subseteq S\} = S^{\downarrow} \cap (S\setminus B(S))^{\uparrow}$. By submodularity of f (Definition 1.1, part 2) the restriction $f_{S^{\le\downarrow}}$ is a monotone nondecreasing function.

Proposition 2.3. *Fix $S \in \mathcal{S}$. Then $f(Y) \ge f^{mon}_{S^{\downarrow}}(Y)$ for all $Y \in S^{\downarrow}$. Moreover, $f(Y) = f^{mon}_{S^{\downarrow}}(Y)$ for all $Y \in S^{\le\downarrow}$.*

Proof. By the definition of $f^{mon}_{S^{\downarrow}}$, we have $f^{mon}_{S^{\downarrow}}(Y) = \min_{Y\subseteq Z\subseteq S} f(Z) \le f(Y)$ for all $Y \in S^{\downarrow}$. Since the restriction $f_{S^{\le\downarrow}}$ is monotone nondecreasing, $f^{mon}_{S^{\downarrow}}(Y) = \min_{Y\subseteq Z\subseteq S} f(Z) = f(Y)$ for all $Y \in S^{\le\downarrow}$. □

The following proposition is implicit in [17]. We give a proof for completeness.

Proposition 2.4. *For all functions $f : 2^{[n]} \to \{0,\ldots,k\}$, the collection of all monotone regions of sets in $\mathcal{S}$ forms a cover of the domain, namely, $\cup_{S\in\mathcal{S}} S^{\le\downarrow} = 2^{[n]}$.*

Proof. The proof is by induction on the value $f([n])$ that the function f takes on the largest set in its domain. The base case of induction is $f([n]) = k$. In this case, $\mathcal{S}$ consists of a single set $S = [n]$, and the function f is monotone non-increasing on $S^{\downarrow} = S^{\le\downarrow}$. Now suppose that the statement holds for all f, such that $f([n]) \ge t$. If $f([n]) = t-1$ then for every $Y \in 2^{[n]}$ there are two cases:

1. There is no Z of size $n-1$ such that $f(Z) > f([n])$ and $Y \in Z^{\downarrow}$ then $Y \in [n]^{\le\downarrow}$ and thus Y is covered by the monotone region of $[n]$.

2. There exists a set Z of size $n-1$ such that $f(Z) > f([n])$ and $Y \in Z^{\downarrow}$ then there exists a set $S \in \mathcal{S}$, such that $Y \in S^{\downarrow}$ by applying inductive hypothesis to $f_{Z^{\downarrow}}$, so Y is covered by $S^{\downarrow}$.

□

Lemma 2.1 and Proposition 2.2 give that the collection of terms C_{mon}, constructed in Line 1 of Algorithm 2, corresponds to a monotone pseudo-Boolean k-DNF representation for $f^{mon}_{S^{\downarrow}}$. By the same argument as in the proof of Lemma 2.1, $|S| \ge n-k$ for all $S \in \mathcal{S}$, since the maximum of f is at most k. Therefore, Line 2 of Algorithm 2 adds at most $n-|S|$ negated variables to every term of C_{mon}, resulting in

terms with at most k positive and at most k negated variables.

It remains to prove that the constructed formula represents f. For a set S, let C_S denote the collection of terms obtained on Line 2 of Algorithm 2. By construction, $C_S(Y) = f^{mon}_{S^\downarrow} \cdot \mathbf{1}_{S^\downarrow}(Y)$ for all $Y \in 2^{[n]}$. For every $Y \in 2^{[n]}$, the first part of Proposition 2.3 implies that $C_S(Y) = f^{mon}_{S^\downarrow}(Y) \cdot \mathbf{1}_{S^\downarrow}(Y) \leq f(Y)$, yielding $\max_{S\in\mathcal{S}} C_S(Y) \leq f(Y)$. On the other hand, by Proposition 2.4, for every $Y \in 2^{[n]}$ there exists a set $S \in \mathcal{S}$, such that $Y \in S^{\leq\downarrow}$. For such S, the second part of Proposition 2.3 implies that $C_S(Y) = f^{mon}_{S^\downarrow}(Y) \cdot \mathbf{1}_{S^\downarrow}(Y) = f(Y)$. Therefore, f is equivalent to $\max_{S\in\mathcal{S}} C_S$. □

3 Generalized switching lemma for pseudo-Boolean DNFs

The following definitions are stated for completeness and can be found in [23, 22].

Definition 3.1 (Decision tree)**.** *A decision tree T is a representation of a function $f\colon \{0,1\}^n \to \mathbb{R}$. It consists of a rooted binary tree in which the internal nodes are labeled by coordinates $i \in [n]$, the outgoing edges of each internal node are labeled 0 and 1, and the leaves are labeled by real numbers. We insist that no coordinate $i \in [n]$ appears more than once on any root-to-leaf path.*

Each input $x \in \{0,1\}^n$ corresponds to a computation path in the tree T from the root to a leaf. When the computation path reaches an internal node labeled by a coordinate $i \in [n]$, we say that T queries x_i. The computation path then follows the outgoing edge labeled by x_i. The output of T (and hence f) on input x is the label of the leaf reached by the computation path. We identify a tree with the function it computes.

The *depth* s of a decision tree T is the maximum length of any root-to-leaf path in T. For a function f, DT-depth(f) is the minimum depth of a decision tree computing f.

Definition 3.2 (Random restriction)**.** *A restriction ρ is a mapping of the input variables to $\{0,1,\star\}$. The function obtained from $f(x_1,\ldots,x_n)$ by applying a restriction ρ is denoted $f|_\rho$. The inputs of $f|_\rho$ are those x_i for which $\rho(x_i) = \star$ while all other variables are set according to ρ.*

A variable x_i is *live* with respect to a restriction ρ if $\rho(x_i) = \star$. The set of live variables with respect to ρ is denoted live(ρ). A random restriction ρ with parameter $p \in (0,1)$ is obtained by setting each x_i, independently, to $0, 1$ or $\star$, so that $\Pr[\rho(x_i) = \star] = p$ and $\Pr[\rho(x_i) = 1] = \Pr[\rho(x_i) = 0] = (1-p)/2$.

We will prove the following generalization of the switching lemma [18, 5].

Lemma 3.1 (Switching lemma for pseudo-Boolean formulas)**.** *Let $f \in DNF^{k,r}$ and ρ be a random restriction with parameter p (i.e., $\Pr[\rho(x_i) = \star] = p$). Then*

$$\Pr[DT\text{-}depth(f|_\rho) \geq s] < r \cdot (7pk)^s.$$

Proof. We use the exposition of Razborov's proof of the switching lemma for Boolean functions, described in [5], as the basis of our proof and highlight the modifications we made for non-Boolean functions.

Define $\mathcal{R}^\ell_n$ to be the set of all restrictions ρ on a domain of n variables that have exactly ℓ unset variables. Fix some function $f \in \mathrm{DNF}^{k,r}$, represented by a formula F, and assume that there is a total order on the terms of F as well as on the indices of the variables. A restriction ρ is applied to F in order, so that F_ρ is a pseudo-Boolean DNF formula whose terms consist of those terms in F that are not falsified by ρ, each shortened by removing any variables that are satisfied by ρ, and taken in the order of occurrences of the original terms on which they are based.

Definition 3.3 (Canonical labeled decision tree)**.** *The* canonical labeled decision tree *for F, denoted $T(F)$, is defined inductively as follows:*

1. *If F is a constant function then $T(F)$ consists of a single leaf node labeled by the appropriate constant.*
2. *If the first term C_1 of F is not empty then let F' be the remainder of F so that $F = \max(C_1, F')$. Let K be the set of variables appearing in C_1.*

The tree $T(F)$ starts with a complete binary tree for K, which queries the variables in K in the order of their indices. Each leaf v_σ in the tree is associated with a restriction σ which sets the variables of K according to the path from the root to v_σ. For each σ, replace the leaf node v_σ by the subtree $T(F_\sigma)$. For the unique assignment σ satisfying C_1, also label the corresponding node by L_σ equal to the maximum of the labels assigned to the predecessors of this node in the tree and the integer constant associated with the term C_1.

Note that the above definition is more general than a canonical decision tree for Boolean DNF formulas because it uses labels for some of the internal nodes in the tree to indicate that the paths from the parent node to these internal nodes restrict the value of the formula to be at least the value of the label. For the Boolean DNFs such labels are not needed and thus if we apply to them the definition above we get a standard definition of a canonical decision tree, as in [5]. Formally, for pseudo-Boolean DNF formulas the label L_σ of the internal node σ represents the fact that the value of the formula on the leaves in the subtree of σ is at least L_σ.

Using the terminology introduced above, we can state the switching lemma as follows.

Lemma 3.2. *Let F be a pseudo-Boolean formula, representing a function $f \in DNF^{k,r}$, $s \geq 0$, $p \leq 1/7$ and $\ell = pn$. Then*

$$\frac{|\{\rho \in \mathcal{R}_n^\ell : |T(F|_\rho)| \geq s\}|}{|\mathcal{R}_n^\ell|} < r(7pk)^s.$$

Proof. Let $stars(k,s)$ be the set of all sequences $\beta = (\beta_1, \ldots, \beta_t)$ such that for each $j \in [t]$, the coordinate $\beta_j \in \{\star, -\}^k \setminus \{-\}^k$ and such that the total number of $\star$'s in all the β_j is s.

Let $S \subseteq \mathcal{R}_n^\ell$ be the set of restrictions ρ such that $|T(F|_\rho)| \geq s$. We will define an injective mapping from S to the cartesian product $\mathcal{R}_n^{\ell-s} \times stars(k,s) \times [2^s] \times [r]$.

Let $F = \max_i C_i$. In the exposition below we use the same notation π to denote both a restriction and a path in the canonical labeled decision tree, which sets variables according to π. Suppose that $\rho \in S$ and π is the restriction associated with the lexicographically first path in $T(F|_\rho)$ of length at least s. Trim the last variables in π along the path π from the root so that $|\pi| = s$. Let c be the maximum label of the node on π (or zero, if none of the nodes on p_π are labeled). Partition the set of terms of F into two sets F' and F'', where F' contains all terms with constants $> c$ and F'' contains all terms with constants $\leq c$ (for Boolean formulas, $c = 0$ and $F = F'$). We will use the subformula F' and π to determine the image of ρ. The image of ρ is defined by following the path π in the canonical labeled decision tree for F_ρ and using the structure of the tree.

Let C_{ν_1} be the first term of F' that is not set to 0 by ρ. Since $|\pi| > 0$, such a term must exist and will not be an empty term (otherwise, the value of $F|_\rho$ is fixed to be $> c$). Let K be the set of variables in $C_{\nu_1}|\rho$ and let σ_1 be the unique restriction of the variables in K that satisfies $C_{\nu_1}|\rho$. Let π_1 be the part of π that sets the variables in K. We have two cases based on whether $\pi_1 = \pi$.

1. If $\pi_1 \neq \pi$ then by the construction of π, restriction π_1 sets all the variables in K. Note that the restriction $\rho\sigma_1$ satisfies the term C_{ν_1} but since $\pi_1 \neq \pi$ the restriction $\rho\pi_1$ does not satisfy term C_{ν_1}.

2. If $\pi_1 = \pi$ then it is possible that π does not set all of the variables in K. In this case we shorten σ_1 to the variables in K that appear in π_1.

Define $\beta_1 \in \{\star, -\}^k$ based on the fixed ordering of the variables in the term C_{ν_1} by letting the jth component of β_1 be $\star$ if and only if the jth variable in C_{ν_1} is set by σ_1. Since $C_{\nu_1}|_\rho$ is not the empty term, β_1 has at least one $\star$. From C_{ν_1} and β_1 we can reconstruct σ_1.

Now by the definition of $T(F|_\rho)$, the restriction $\pi \setminus \pi_1$ labels a path in the canonical labeled decision tree $T(F|_{\rho\pi_1})$. If $\pi_1 \neq \pi$, we repeat the argument above, replacing π and ρ with $\pi \setminus \pi_1$ and $\rho\pi_1$, respectively, and find a term C_{ν_2} which is the first term of F' not set to 0 by $\rho\pi_1$. Based on this, we generate π_2, σ_2 and β_2, as before. We repeat this process until the round t in which $\pi_1\pi_2 \ldots \pi_t = \pi$.

Let $\sigma = \sigma_1\sigma_2\ldots\sigma_t$. We define $\xi \in \{0,1\}^s$ to be a vector that indicates for each variable set by π whether it is set to the same value as σ sets it. We define the image of ρ in the injective mapping as a quadruple, $\langle \rho\sigma_1\ldots\sigma_t, (\beta_1,\ldots,\beta_t), \xi, c\rangle$. Because $\rho\sigma \in \mathcal{R}_n^{\ell-s}$ and $(\beta_1,\ldots,\beta_t) \in stars(k,s)$ the mapping is as described above.

It remains to show that the defined mapping is indeed injective. We will show how to invert it by reconstructing ρ from its image. We use c to construct F' from F. The reconstruction procedure is iterative. In one stage of the reconstruction we recover $\pi_1\ldots\pi_{i_1}, \sigma_1\ldots\sigma_{i-1}$ and construct $\rho\pi_1\ldots\pi_{i-1}\sigma_i\ldots\sigma_t$. Recall that for $i < t$ the restriction $\rho\pi_1\ldots\pi_{i-1}\sigma_i$ satisfies the term C_{ν_i}, but does not satisfy terms C_j for all $j < \nu_i$. This holds if we extend the restriction by appending $\sigma_{i+1}\ldots\sigma_t$. Thus, we can recover ν_i as the index of the first term of F' that is not falsified by $\rho\pi_1\ldots\pi_{i-1}\sigma_i\ldots\sigma_t$ and the consant corresponding to this term is at least c.

Now, based on C_{ν_1} and β_i, we can determine σ_i. Since we know $\sigma_1\ldots\sigma_i$, using the vector ξ we can determine π_i. We can now change $\rho\pi_1\ldots\pi_{i-1}\sigma_i\ldots\sigma_t$ to $\rho\pi_1\ldots\pi_{i-1}\pi_i\sigma_{i+1}\ldots\sigma_t$ using the knowledge of π_i and σ_i. Finally, given all the values of the π_i we reconstruct ρ by removing the variables from $\pi_1\ldots\pi_t$ from the restriction.

The computation in the following claim completes the proof of the switching lemma. □

□

Claim 1 ([5]). *For $p < 1/7$ and $p = \ell/n$, the following holds:*

$$\frac{|\mathcal{R}_n^{\ell-s}| \cdot |stars(k,s)| \cdot 2^s}{|\mathcal{R}_n^{\ell}|} < (7pk)^s.$$

Proof. We have $|\mathcal{R}_n^{\ell}| = \binom{n}{\ell}2^{n-\ell}$, so:

$$\frac{|\mathcal{R}_n^{\ell-s}|}{|\mathcal{R}_n^{\ell}|} \le \frac{(2\ell)^s}{(n-\ell)^s}.$$

We use the following bound on $|stars(k,s)|$.

Proposition 3.3 (Lemma 2 in [5]). $|stars(k,s)| < (k/\ln 2)^s$.

Using Proposition 3.3 we get:

$$\begin{aligned}\frac{|S|}{|\mathcal{R}_n^{\ell}|} &\le \frac{|\mathcal{R}_n^{\ell-s}|}{|\mathcal{R}_n^{\ell}|} \cdot |stars(k,s)| \cdot 2^s \\ &\le \left(\frac{4\ell k}{(n-\ell)\ln 2}\right)^s \\ &= \left(\frac{4pk}{(1-p)\ln 2}\right)^s.\end{aligned}$$

For $p < 1/7$, the last expression is at most $(7pk)^s$, as claimed. □

4 Learning pseudo-Boolean DNFs

In this section, we present our learning results for pseudo-Boolean k-DNF and prove Theorem 1.2.

Let R_r denote the set of multiples of $2/(r-1)$ in the interval $[-1,1]$, namely $R_r = \{-1, -1 + 2/(r-1), ..., 1 - 2/(r-1), 1\}$. First, we apply a transformation of the range by mapping $\{0,\ldots,r\}$ to R_r. Formally, in this section instead of functions $f\colon \{0,1\}^n \to \{0,\ldots,r\}$ we consider functions $f'\colon \{-1,1\}^d \to [-1,1]$, such that $f'(x'_1,\ldots,x'_n) = 2/(r-1)\cdot f(x_1,\ldots,x_n) - 1$, where $x'_i = 1 - 2x_i$. The mapping is one to one and thus a learning algorithm for the class of functions that can be represented by pseudo-Boolean DNF formulas of width k with constants in the range R_r implies Theorem 1.2. Thus, we will refer to this transformed class also as $\mathrm{DNF}^{k,r}$.

For a set $S \subseteq [n]$, let χ_S be the standard Fourier basis vector and let $\hat{f}(S)$ denote the corresponding Fourier coefficient of a function f.

Definition 4.1. *A function g ϵ-approximates a function f if $\mathbb{E}[(f-g)^2] \le \epsilon$. A function is M-sparse if it has at most M non-zero Fourier coefficients. The Fourier degree of a function, denoted $deg(f)$, is the size of the largest set, such that $\hat{f}(S) \neq 0$.*

The following guarantee about approximation of functions in $\mathrm{DNF}^{k,r}$ by sparse functions is the key lemma in the proof of Theorem 1.2.

Theorem 4.1. *Every function $f \in DNF^{k,r}$ can be ϵ-approximated by an M-sparse function, where $M = k^{O(k\log(r/\epsilon))}$.*

Proof of Theorem 4.1. We generalize the proof by Mansour [22], which relies on multiple applications of the switching lemma. Our generalization of the switching lemma allows us to obtain the following parameters of the key statements in the proof, which bound the L_2-norm of the Fourier coefficients of large sets in Lemma 4.2 and the L_1-norm of the Fourier coefficients of small sets in Lemma 4.4.

Lemma 4.2. *For every function $f \in DNF^{k,r}$,*

$$\sum_{S\colon |S|>28k\log(2r/\epsilon)} \hat{f}^2(S) \leq \epsilon/2.$$

Proof. The proof relies on the following result.

Proposition 4.3 ([22, 23])**.** *Let $f\colon \{0,1\}^n \to \{-1,1\}$ and f_ρ be a random restriction with parameter p. Then for every $t \in [n]$,*

$$\sum_{|S|>t} \hat{f}^2(S) \leq \Pr_\rho[deg(f|_\rho) \geq tp/2].$$

Because $deg(f|_\rho) \leq$ DT-depth$(f|_\rho)$ and thus $\Pr[deg(f|_\rho) \geq tp/2] \leq \Pr[\text{DT-depth}(f|_\rho) \geq tp/2]$. By using Lemma 3.1 and setting $p = 1/14k$ and $t = 28k\log(2r/\epsilon)$, we complete the proof. □

The main part of the proof of Theorem 4.1 is the following lemma, which bounds the L_1-norm of Fourier coefficients, corresponding to sets of bounded size.

Lemma 4.4. *For every function $f \in DNF^{k,r}$ and $\tau \in [n]$,*

$$\sum_{S\colon |S|\leq\tau} |\hat{f}(S)| \leq 4r(28k)^\tau = rk^{O(\tau)}.$$

Proof. Let $L_{1,t}(f) = \sum_{|S|=t} |\hat{f}(S)|$ and $L_1(f) = \sum_{t=0}^n L_{1,t}(f) = \sum_S |\hat{f}(S)|$.

We use the following bound on $L_1(f)$ for decision trees.

Proposition 4.5 ([20, 23])**.** *Consider a function $f\colon \{-1,1\}^n \to [-1,1]$, such that DT-depth$(f) \leq s$. Then $L_1(f) \leq 2^s$.*

We show the following generalization of Lemma 5.2 in [22] for DNF^{k,r}.

Proposition 4.6. *Let $f \in DNF^{k,r}$ and let ρ be a random restriction of f with parameter $p \leq 1/28k$. Then $\mathbb{E}_\rho[L_1(f|_\rho)] \leq 2r$.*

Proof. By the definition of expectation,

$$\begin{aligned}&\mathbb{E}_\rho[L_1(f)]\\ =&\sum_{s=0}^n \Pr[\text{DT-depth}f|_\rho = s]\\ &\cdot\mathbb{E}_\rho\left[L_1(f|_\rho) \mid \text{DT-depth}(f|_\rho) = s\right].\end{aligned}$$

By Proposition 4.5, for all ρ, such that DT-depth$(f|_\rho) = s$, it holds that $L_1(f) \leq 2^s$. By Lemma 3.1, $\Pr[\text{DT-depth}(f|_\rho) \geq s] \leq r(7pk)^s$. Therefore, $\mathbb{E}_\rho[L_1(f)] \leq \sum_{s=0}^n r(7pk)^s 2^s = r\cdot\sum_{s=0}^n (14pk)^s$. For $p \leq 1/28k$ the lemma follows. □

We use Lemma 5.3 from [22] to bound $L_{1,t}(f)$ by the value of $\mathbb{E}_\rho[L_{1,t}(f|_\rho)]$. Because in [22] the lemma is stated for Boolean functions, we give the proof for real-valued functions for completeness.

Proposition 4.7 ([22])**.** *For $f\colon \{0,1\}^n \to [-1,1]$ and a random restriction ρ with parameter p,*

$$L_{1,t}(f) \leq \left(\frac{1}{p}\right)^t \mathbb{E}_\rho\left[L_{1,t}(f)\right].$$

Proof. Consider a random variable $\mathcal{L}$ supported on $2^{[n]}$, such that for each x_i independently, $\Pr[x_i \in \mathcal{L}] = p$. The random variable $\mathcal{L}$ is the set of live variables in a random restriction with parameter p. We can rewrite $L_{1,t}$ as:

$$\begin{aligned}L_{1,t}(f) &= \sum_{|S|=t} |\hat{f}(S)|\\ &= \left(\frac{1}{p}\right)^t \mathbb{E}_{\mathcal{L}}\left[\sum_{S\subseteq\mathcal{L},|S|=t} \left|\hat{f}(S)\right|\right]. \qquad (2)\end{aligned}$$

For an arbitrary choice of $\mathcal{L}$ and a subset $S \subseteq \mathcal{L}$ we have:

$$\begin{aligned}|\hat{f}(S)| &= |\mathbb{E}_{x_1,\ldots,x_n}\left[f(x_1,\ldots,x_n)\chi_S(x_1,\ldots,x_n)\right]|\\ &\leq \mathbb{E}_{x\notin\mathcal{L}}|\mathbb{E}_{x\in\mathcal{L}}\left[f(x_1,\ldots,x_n)\chi_S(x_1,\ldots,x_n)\right]|\\ &= \mathbb{E}_\rho\left[|\hat{f}|_\rho(S)| \mid live(\rho) = \mathcal{L}\right],\end{aligned}$$

where the last line follows from the observation that averaging over $x_i \notin \mathcal{L}$ is the same as taking the expectation of a random restriction whose set of live variables is restricted to be $\mathcal{L}$. Because the absolute value of every coefficient S is expected to increase, this implies that:

$$\begin{aligned}&\sum_{S\subseteq\mathcal{L}}\left|\hat{f}(S)\right|\\ \leq&\mathbb{E}_\rho\left[\sum_{S\subseteq\mathcal{L},|S|=t}|\hat{f}|_\rho(S)|\mid live(\rho)=\mathcal{L}\mid\right]\\ =&\mathbb{E}_\rho\left[L_{1,t}(f_\rho)|live(\rho)=\mathcal{L}\right].\end{aligned}$$

Using this together with (2), we conclude that

$$\begin{aligned}&L_{1,t}(f)\\ =&\left(\frac{1}{p}\right)^t\mathbb{E}_{\mathcal{L}}\left[\sum_{S\subseteq\mathcal{L},|S|=t}\left|\hat{f}(S)\right|\right]\\ \leq&\left(\frac{1}{p}\right)^t\mathbb{E}_\rho\left[L_{1,t}(f|_\rho)\right].\end{aligned}$$

□

Note that $\sum_{S:\,|S|\leq\tau}|\hat{f}(S)| = \sum_{t=0}^{\tau}L_{1,t}(f)$. By setting $p = 1/28k$ and using Propositions 4.6 and 4.7, we get:

$$L_{1,t}(f) \leq 2r(28k)^t.$$

Thus, $\sum_{S:\,|S|\leq\tau}|\hat{f}(S)| \leq 4r(28k)^\tau = rk^{O(\tau)}$, completing the proof of Lemma 4.4. □

Let $\tau = 28k\log(2r/\epsilon)$ and $L = \sum_{|S|\leq\tau}|\hat{f}(S)|$. Let $G = \{S : |\hat{f}(S)| \geq \epsilon/2L \text{ and } |S| \leq \tau\}$ and $g(x) = \sum_{S\in G}\hat{f}(S)\chi_S(x)$. We will show that g is M-sparse and that it ϵ-approximates f.

By an averaging argument, $|G| \leq 2L^2/\epsilon$. Thus, function g is M-sparse, where $M \leq 2L^2/\epsilon$. By Lemma 4.4, $L = rk^{O(\tau)} = k^{O(k\log(r/\epsilon))}$. Thus, $M = k^{O(k\log(r/\epsilon))}$, as claimed in the theorem statement.

By the definition of g and by Parseval's identity,

$$\begin{aligned}&\mathbb{E}[(f-g)^2]\\ =&\sum_{S\notin G}\hat{f}^2(S)\\ =&\sum_{S:\,|S|>\tau}\hat{f}^2(S)+\sum_{S:\,|S|\leq\tau,|\hat{f}(S)|\leq\epsilon/2L}\hat{f}^2(S).\end{aligned}$$

By Lemma 4.2, the first summation is at most $\epsilon/2$. For the second summation, we get:

$$\begin{aligned}&\sum_{S:\,|S|\leq\tau,|\hat{f}(S)|\leq\epsilon/2L}\hat{f}^2(S)\\ \leq&\left(\max_{S:\,|\hat{f}(S)|\leq\epsilon/2L}|\hat{f}(S)|\right)\left(\sum_{|S|\leq\tau}|\hat{f}(S)|\right)\\ \leq&\frac{\epsilon}{2L}\cdot L=\epsilon/2.\end{aligned}$$

This implies that $\mathbb{E}[(f-g)^2] \leq \epsilon$ and thus g ϵ-approximates f. □

To get a learning algorithm and prove Theorem 1.2 we can use the sparse approximation guarantee of Theorem 4.1 together with Kushilevitz-Mansour learning algorithm (for PAC-learning) or the learning algorithm of Gopalan, Kalai and Klivans (for agnostic learning).

Proof of Theorem 1.2. We will use the learning algorithm of Kushilevitz and Mansour [15, 19], which gives the following guarantee:

Theorem 4.8 ([19])**.** *Let f be a function that can be ϵ-approximated by an M-sparse function. There exists a randomized algorithm, whose running time is polynomial in M, n, $1/\epsilon$ and $\log(1/\delta)$, that given oracle access to f and $\delta > 0$, with probability at least $1-\delta$ outputs a function h that $O(\epsilon)$-approximates f.*

Setting the approximation parameter in Theorem 4.8 to be $\epsilon' = \epsilon/Cr^2$ for large enough constant C and taking $M = k^{O(k\log(r/\epsilon'))}$ we get an algorithm which returns a function h that (ϵ/r^2)-approximates f. The running time of such algorithm is polynomial in n, $k^{O(k\log(r/\epsilon))}$ and $\log(1/\delta)$. By Proposition 4.9,

if we round the values of h in every point to the nearest multiple of $2/(r-1)$, we will get a function h', such that $\Pr_{x\in U^n}[h'(x) \neq f(x)] \leq \epsilon$, completing the proof.

Proposition 4.9. *Suppose a function $h : 2^{[n]} \to [-1,1]$ is an ϵ-approximation for $f\colon 2^{[n]} \to R_r$. Let g be the function defined by $g(x) = argmin_{y\in R_r}|h(x)-y|$, breaking ties arbitrarily. Then $\Pr_{x\in U^n}[g(x) \neq f(x)] \leq \epsilon\cdot(r-1)^2$.*

Proof of Proposition 4.9. Observe that $|f(x)-h(x)|^2 \geq 1/(r-1)^2$ whenever $f(x) \neq g(x)$. This implies

$$\begin{aligned}\Pr_{x\in U^n}[g(x)\neq f(x)] \leq \\ \Pr_{x\in U^n}[(r-1)^2\cdot|f(x)-h(x)|^2 \geq 1] \leq \\ \mathbb{E}_{x\in U^n}[(r-1)^2\cdot|f(x)-h(x)|^2] \leq \\ (r-1)^2\cdot\mathbb{E}_{x\in U^n}[|f(x)-h(x)|^2] \leq \epsilon(r-1)^2.\end{aligned}$$

The last inequality follows from the definition of ϵ-approximation. □

Extension of our learning algorithm to the agnostic setting follows from the result of Gopalan, Kalai and Klivans.

Theorem 4.10 ([16])**.** *If every function f in a class C has an M-sparse ϵ-approximation, then there is an agnostic learning algorithm for C with running time $poly(n, M, 1/\epsilon)$.*

This completes the proof of Theorem 1.2. □

Acknowledgments

We are grateful to Jan Vondrak, Vitaly Feldman, Lev Reyzin, Nick Harvey, Paul Beame, Ryan O'Donnell and other people for their feedback and comments on the results in this paper.

References

[1] A. Badanidiyuru, S. Dobzinski, H. Fu, R. Kleinberg, N. Nisan, and T. Roughgarden. Sketching valuation functions. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 1025–1035. SIAM, 2012.

[2] M.-F. Balcan, F. Constantin, S. Iwata, and L. Wang. Learning valuation functions. *CoRR*, abs/1108.5669, 2011.

[3] M.-F. Balcan and N. J. A. Harvey. Learning submodular functions. *CoRR*, abs/1008.2159, 2010.

[4] M.-F. Balcan and N. J. A. Harvey. Learning submodular functions. In *STOC*, pages 793–802, 2011.

[5] P. Beame. A switching lemma primer. In *Unpublished notes: http://www.cs.washington.edu/homes/beame/papers/primer.ps*, 1994.

[6] E. Blais, J. Brody, and K. Matulef. Property testing lower bounds via communication complexity. In *IEEE Conference on Computational Complexity*, pages 210–220, 2011.

[7] S. Boucheron, G. Lugosi, and P. Massart. A sharp concentration inequality with application. *Random Struct. Algorithms*, 16:277–292, May 2000.

[8] S. Boucheron, G. Lugosi, and P. Massart. On concentration of self-bounding functions. *Electron. J. Probab.*, 14:1884–1899, 2009.

[9] D. Chakrabarty and Z. Huang. Testing coverage functions. *CoRR*, abs/1205.1587, 2012. Accepted to ICALP 2012.

[10] M. Cheraghchi, A. Klivans, P. Kothari, and H. K. Lee. Submodular functions are noise stable. In *SODA*, pages 1586–1592, 2012.

[11] Y. Crama and P. L. Hammer. *Boolean Functions—Theory, Algorithms, and Applications*, volume 142 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2011.

[12] O. Ekin, P. L. Hammer, and U. N. Peled. Horn functions and submodular boolean functions. *Theor. Comput. Sci.*, 175(2):257–270, 1997.

[13] M. X. Goemans, N. J. A. Harvey, S. Iwata, and V. S. Mirrokni. Approximating submodular functions everywhere. In *SODA*, pages 535–544, 2009.

[14] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45(4):653–750, 1998.

[15] O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *STOC*, pages 25–32, 1989.

[16] P. Gopalan, A. T. Kalai, and A. R. Klivans. Agnostically learning decision trees. In *STOC*, pages 527–536, 2008.

[17] A. Gupta, M. Hardt, A. Roth, and J. Ullman. Privately releasing conjunctions and the statistical query barrier. In *STOC*, pages 803–812, 2011.

[18] J. Håstad. Almost optimal lower bounds for small depth circuits. In *STOC*, pages 6–20, 1986.

[19] E. Kushilevitz and Y. Mansour. Learning decision trees using the fourier sprectrum (extended abstract). In *STOC*, pages 455–464, 1991.

[20] E. Kushilevitz and Y. Mansour. Learning decision trees using the Fourier spectrum. *SIAM J. Comput.*, 22(6):1331–1348, 1993.

[21] L. Lovasz. Submodular functions and convexity. In *Mathematical Programming and the State of the Art*, pages 233–257, 1982.

[22] Y. Mansour. An $O(n^{\log\log n})$ learning algorithm for DNF under the uniform distribution. *J. Comput. Syst. Sci.*, 50(3):543–550, 1995.

[23] R. O'Donnell. *Analysis of Boolean Functions (http://analysisofbooleanfunctions.org)*. 2012.

[24] M. Parnas, D. Ron, and R. Rubinfeld. On testing convexity and submodularity. *SIAM J. Comput.*, 32(5):1158–1184, 2003.

[25] M. Parnas, D. Ron, and A. Samorodnitsky. Testing basic boolean formulae. *SIAM J. Discrete Math.*, 16(1):20–46, 2002.

[26] R. Rubinfeld and M. Sudan. Robust characterization of polynomials with applications to program testing. *SIAM J. Comput.*, 25(2):252–271, 1996.

[27] C. Seshadhri. Question 2: Testing submodularity. In P. Indyk, A. McGregor, I. Newman, and K. Onak, editors, *Open Problems in Data Streams, Property Testing, and related topics, Bertinoro Workshop on Sublinear Algorithms (May 2011) and IITK Workshop on Algorithms for Processing Massive Data Sets (December 2009)*, page 3, 2011. Downloaded July 2, 2012 from `http://people.cs.umass.edu/~mcgregor/papers/11-openproblems.pdf`.

[28] C. Seshadhri and J. Vondrák. Is submodularity testable? In *ICS*, pages 195–210, 2011.

[29] L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984.

[30] J. Vondrák. A note on concentration of submodular functions. *CoRR*, abs/1005.2791, 2010.

A Converting a learner into a proper learner

Let $\mathcal{C}$ be a class of discrete objects represented by functions over a domain of "size" n.

Proposition A.1. *If there exists a learning algorithm L for a class $\mathcal{C}$ with query complexity $q(n,\epsilon)$ and running time $t(n,\epsilon)$, then there exists a proper learning algorithm L' for $\mathcal{C}$ with query complexity $q(n,\epsilon/2)$ and running time $t(n,\epsilon/2)+|\mathcal{C}|$.*

Proof. Given parameters n,ϵ and oracle access to a function f, the algorithm L' first runs L with parameters $n,\epsilon/2$ to obtain a hypothesis g. Then it finds and outputs a function $h\in\mathcal{C}$, which is closest to g, namely $h = argmin_{h'\in\mathcal{C}} dist(g,h')$. By our assumption that L is a learning algorithm, $dist(f,g)\le\epsilon/2$. Since $f\in\mathcal{C}$, we have $dist(g,h)\le dist(g,f)\le\epsilon/2$. By the triangle inequality, $dist(f,h)\le dist(f,g)+dist(g,h)\le\epsilon$. □